*Exercise 2: Const-ness (constness.cc)*

***Explained***

* **This exercise had us explore the differences between const variables/arguments/parameters and non-const variables/arguments/parameters.**
* **The first code block worked as expected:**
  + Print 10 equal signs for aesthetic separation.
  + string ncs = "non-const string"
  + &ncs = 0x7ffe5345a800 (The address of the variable)
  + non\_const\_by\_ref(ncs); Prints out address (0x7ffe5345a800), size of string (16 chars including whitespace in between words), then successfully appends another string to the original string as expected.
  + There is no return statement so now all variables created in this function block will be destroyed as we have reached the end, **BUT** because we have passed by reference the original string has been successfully amended.
  + New string = non\_const\_by\_ref(ncs) = "non-const string has been modified"
* **Second code block...**
  + Again, print 10 equal signs for aesthetic separation.
  + Since variables are destroyed after a code block, now we must re-instantiate our variables.
  + string ncs = "non-const string"
  + &ncs = 0x7ffe5345a800 (it's weird to me that the address is still the same. Technically, since the variables from the previous code block were destroyed, shouldn’t this be a new address? I tested this below in a separate file and it was a new address…)
  + non\_const\_by\_val(ncs); Prints out address (0x7ffe5345a880), size of string (16 chars including whitespace in between words), then successfully appends another string to the original string.
  + Notice that the address is different (this time expected behavior) from inside the main code block and from inside the function call. This is because, when passing by value, we are passing a copy. A copy looks the same, but is stored at a new address.
  + Again, there is no return statement. The difference this time is that the append was successful, but because we passed by value a mere copy was sent to this function. The copy was successfully appended to, but not the original. Therefore, with no return statement all changes are destroyed. This is why when print out the value of “ncs after” back in the original code block from main, the string still returns the original value without the append.
  + New string = non\_const\_by\_val(ncs) = "non-const string"

Graphical user interface, text

Description automatically generated

*Address from different code blocks are different here as expected.*

*Text

Description automatically generated*

*Yet address from two different code blocks (before passing it to the function) are the same…*

* **Constant code blocks 3 & 4:**
  + These are the same as 1 & 2 except that the const keyword has been added to the function header in both (respectively), and that the “.append()” line has been commented out in both.
  + Running with the “.append()” commented out successfully produces the same behavior as above. The only differences in the output are the addresses when passed by value vs when passed by ref as explained above.
  + Now when trying to add the “.append()” line back to either or both functions results in an explosion… (an error rather).
  + It seems that, regardless of whether the argument is passed by value or reference, it is trumped by the const keyword which won’t allow any modifications to the argument.

*Text

Description automatically generated*

* **Code block 5:**
  + Now it seems we are experimenting to see which overloaded function will get called.
  + We have two overloaded functions, the only dif being const in the header:
    - overloaded\_func(string & s)
    - overloaded\_func(string const & s)
  + Again print 10 “=” for separation.
  + Again, re-instantiate string ncs = "non-const string".
  + Now pass “ncs” to overloaded\_func(ncs);
  + *\*Jeopardy theme music plays\**
  + Our output says: “overloaded\_func() with non-const ref called on non-const string”. This def makes sense…
  + Just for fun, lets see what happens if we pass a const string as the argument:

*Text

Description automatically generated*

***Yup, passing in a constant string causes the const overloaded function to be called!***

* **Code block 6 – 10 (Constant string experiments):**
  + Here we are doing the same thing as in code blocks 1-5, except now we are testing a const string as our argument to see if/how it’ll behave differently in all of the previous situations.
* **Code block 6:**
  + When uncommenting “LINE 3” we get another crash because we are trying to pass a const argument into a function that only accepts parameters by reference. By the literal definition this is inherently impossible, since passing something by reference assumes that the original value will be manipulated. I guess because of this assumption (even though we may not actually manipulate anything inside the function block), the compiler won’t even allow this to happen.

***Text

Description automatically generated***

* **Code block 7 (Passing a const argument to a “non\_const\_by\_val()” function) :**
  + This one works because event though the original argument is an immutable constant, once it hits the function it is copied (because it is passed by value) and therefore can be manipulated.
  + Keep in mind that though that for this very same reason, the final string (called from code block 7 in main) does not change simply because all the changes we made to the copy while in the function call get destroyed once the function is complete.
* **Code block 8 & 9 (Passing a const argument to a “const\_by\_ref()” and a “const\_by\_val()” function respectively) :**
  + Again, these two functions only work if the “.append()” lines are commented out. Why? Because we cannot append to a const.
  + **Code block 10 (Tested above during code block 5)**